STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN TOIMIALA

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

WEB-POHJAISEN VIDEOEDITORIN TOTEUTUS JA OBJECT-RELATIONAL MAPPING

Työn tekijä: Matti Venäläinen
Työn valvoja: Matti Luukkainen
Työn ohjaaja: Juhana Kokkonen

Työ hyväksytty: __. __. 2007

Matti Luukkainen
yliopettaja

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ALKULAUSE

Tämä insinöörityö tehtiin Helsingin ammattikorkeakoulun verkkoviestinnälle. Haluan kiittää projektin parissa työskenteleviä osapuolia tuottoisasta ja innovatiivisesta yhteistyöstä. Toivon projektille menestystä tuleville vuosille kuten myös tuleville tämän projektin parissa työskenteleville insinööriopiskelijoille sekä verkkoviestinnän opiskelijoille. Haluan erityisesti kiittää työn valvojaa yliopettaja Matti Luukkaista sekä työn ohjaajaa lehtori Juhana Kokkosta sallimalla liikkumavaraa ja tarjoamalla työskentelymahdollisuudet tässä projektissa. Viimeisenä haluan kiittää työskentelypariani ja opiskelijatoveriani Markku Lempistä, jonka kanssa työskentely mahdollisti tämän insinöörityön valmistumisen.

Vantaalla 9.4.2007

Matti Venäläinen

STADIA
HELSINGIN AMMATTIKORKEAKOULU

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Matti Venäläinen

Työn nimi: Web-pohjaisen videoeditorin toteutus ja Object-Relational Mapping

Päivämäärä: 9.4.2007 Sivumäärä: 41 s. + 0 liitettä

Koulutusohjelma: Tietotekniikka Suuntautumisvaihtoehto: Ohjelmistotekniikka

Työn valvoja: yliopettaja Matti Luukkainen

Työn ohjaaja: lehtori Juhana Kokkonen

Tässä insinöörityössä esitellään Stadian verkkoviestinnän VIDEOS-hankkeeseen liittyvän web-pohjaisen videoeditorin kehitys ja käytetyt teknologiat. Fooga-nimiseksi nimetty videoeditorin käyttämät tekniikat ovat Ruby, Ruby on Rails, FFmpeg, Mencoder, ImageMagick ja FLVTool2. Ruby on olio-pohjainen skriptikieli, Ruby on Rails on web-sovelluskehys ja muut tekniikat ovat komentorivipohjaisia työkaluja, jotka tarjoavat tärkeimmät toiminnallisuudet Foogalle. Tavoitteina oli tämän työn yhteydessä ohjelmoida Foogaan perustoiminnallisuudet, jotka mahdollistavat minimaaliset käyttömahdollisuudet kevääseen 2007 mennessä. Kehitystyö jatkuu vuoteen 2009 asti tarjoamalla samalla mahdollisuuden usealle insinöörityölle tekniikan ja liikenteen koulutusohjelmasta. Tämän lisäksi tässä insinöörityössä perehdytään Object-Relational Mapping-tekniikan perusteisiin ja verrataan Ruby on Railsin ja Javan ORM-ominaisuuksia. Ruby on Railsin osalta esitellään ActiveRecord-luokka ja Javan osalta Hibernate, jonka johdantona on DAO/DTO-sunnittelumalli.

Avainsanat: Ruby, Ruby on Rails, Fooga, VIDEOS, Object-Relational Mapping, Hibernate, Active Record

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ABSTRACT

Name: Matti Venäläinen

Title: A web-based video editor's development and Object-Relational Mapping

Date: 9.4.2007 Number of pages: 41

Department: Information Technology Study Programme: Software Engineering

Instructor: Matti Luukkainen

Supervisor: Juhana Kokkonen

This final thesis presents the development of Helsinki Polytechnic's VIDEOS-programme's web-based video editor and the technologies involved. The aim of this work was to provide the editor with basic functionality, setting the basis on which the future developers will work until the end of 2009. The video editor was named Fooga and it is based on Ruby on Rails web application framework and relies on four different command line tools which provide the essential features. This work reviews the basics of Object-Relational Mapping and includes a comparison of the ORM-properties of Ruby on Rails (Active Record) and Java (Hibernate), as well as a brief introduction of the DAO/DTO model.

Keywords: Ruby, Ruby on Rails, Fooga, VIDEOS, Object-Relational Mapping, Hibernate, Active Record

SISÄLLYS

ALKULAUSE

TIIVISTELMÄ

ABSTRACT

SISÄLLYS

1 JOHDANTO

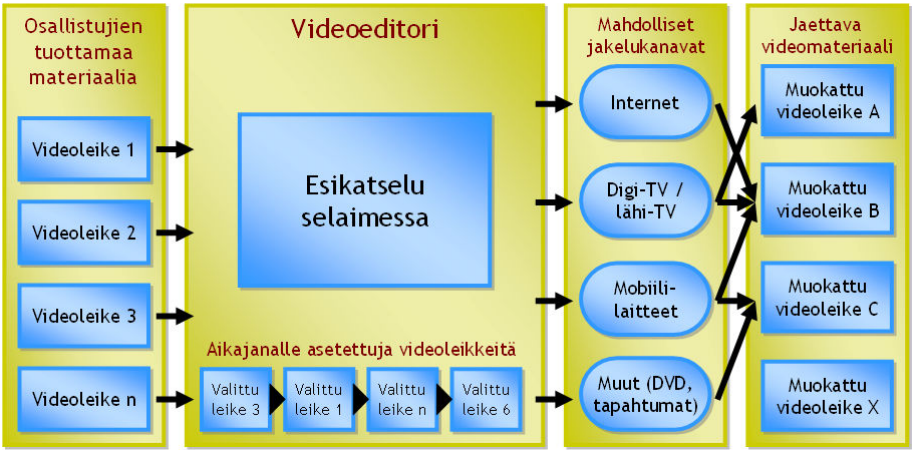
Tämä insinöörityö liittyy Helsingin ammattikorkeakoulu Stadian verkkoviestinnän VIDEOS-hankkeeseen [1]. VIDEOS-hankkeeseen liittyvään projektityöskentelyyn kuuluu ohjelmistokehitys, jossa on tehtävänä suunnitella ja toteuttaa yhteisöllinen www-pohjainen videoeditori ja projektityöskentelyväline. Projektin aikana toteutettava sovellus tullaan julkaisemaan jollain avoimen lähdekoodin lisenssillä.

Sovellus hyödyntää valmiiden komentorivisovellusten tarjoamia palveluita videoiden muokkaamiseen ja lisää niiden päälle käyttäjäystävällisen käyttöliittymän ja tarjoaa käyttäjilleen työvälineet yhteisölliseen toimintaan. Videoeditorisovelluksen nimeksi annettiin Fooga sen fuuga-sävellysmuotoa muistuttavan käyttöperiaatteen mukaisesti: muiden tekemiä videoita voidaan käyttää materiaalina omissa tuotannoissa, joita muut käyttäjät voivat käyttää edelleen omissa projekteissaan. Foogan käyttöliittymä on tarkoitus tehdä mahdollisimman helpoksi käyttää. Samalla se muistuttaa perinteisten videoeditorien käyttöliittymiä mahdollisimman todenmukaisesti.

Foogan alustava suunnittelutyö aloitettiin syksyllä 2006 ja tammikuun 2007 alussa aloitettiin virallinen kehitystyö. Tässä työssä raportoidaan Fooga-projektin kehitys 2.1.2007 ja 15.4.2007 välisenä aikana, jolloin tavoitteena on toteuttaa ensimmäinen toimiva versio rajatuilla perustoiminnallisuuksilla ja raportin pääpaino asetetaan palvelinpuolen toimintaan. Projektin alussa ohjelmistotuotantomenetelmäksi valittiin ketterät menetelmät (sivu 8) , jotta pienelle toteutusryhmälle saataisiin mahdollisimman paljon liikkumavaraa ja että muuttuviin vaatimusmäärittelyihin voitaisiin reagoida mahdollisimman ripeästi. Ketterien menetelmien yleisiä periaatteita noutatetaan löyhästi, mitään nimettyä menetelmää ei käytetä.

Tämän työn tavoitteet

Tämän työn tavoittena on Foogan palvelinpuolen perustoiminnallisuuksien toteuttaminen. Editorin perustoimintoihin sisältyvät videoiden lähettäminen palvelimelle sekä tavalliselta verkkoon liitetyltä tietokoneelta että xhtml-selaimella varustetulta mobiililaitteelta, videoiden tuominen projektiin, yksittäisten videoiden katselu sekä videoleikkeitä muokatessa alku- ja loppukohtien asettaminen, videoleikkeiden järjestäminen projektin aikajanalla sekä aikajanan mukaisen kokonaisen videotiedoston tuottaminen ja lataaminen omalle koneelle (kuva 1).



Kuva 1. Foogan toimintaperiaate [1]

Tavoitteeksi asetettiin myös, että sovelluksen muokkaaminen Foogan ylläpitäjän omien tarpeiden mukaiseksi olisi mahdollisimman yksinkertaista. Muokattavuusnäkökulman mukaisesti pyrittiin välttämään kovakoodattuja tekstejä ja arvoja niin pitkälti, kuin se oli järkevää. Foogan lokalisointia eli käytännössä käyttöliittymän tekstien erikielisiä toteutuksia silmälläpitäen kehitettiin CSV-tiedostoa (Comma Separated Values) käyttävä toteutus. CSV-tiedostoa voi muokata esimerkiksi taulukkolaskentaohjelmalla siten, että uuden sarakkeen lisääminen ja sen rivien täyttäminen riittää uuden kielitoteutuksen lisäämiselle Fooga-järjestelmään. Foogan asetusten

konfigurointi päätettiin suorittaa YAML-tiedostoja (YAML Ain't Markup Language) käyttäen.

2 PROJEKTIN KUVAUS

Projektin alussa päätettiin, että Foogan käyttöliittymä tullaan toteuttamaan pääosin JavaScriptillä ja palvelinpuoli Ruby on Rails:illä (RoR). Koska projektin on tarjoitus tuottaa avoimen lähdekoodin ohjelmisto, päätettiin, että ulkopuolisia Flash-komponentteja käytettäisiin mahdollisimman vähän ja että käyttöliittymä pohjautuisi vankasti JavaScriptiin (JS) ja AJAXiin (Asynchronous JavaScript and XML). JavaScriptin käytöllä käyttöliittymään saataisiin lisättyä dynaamisuutta. Todettiin, että ainoa ehdottomasti välttämätön Flash-komponentti on videosoitin, jolla FLV-tiedostot saadaan näkymään käyttäjälle.

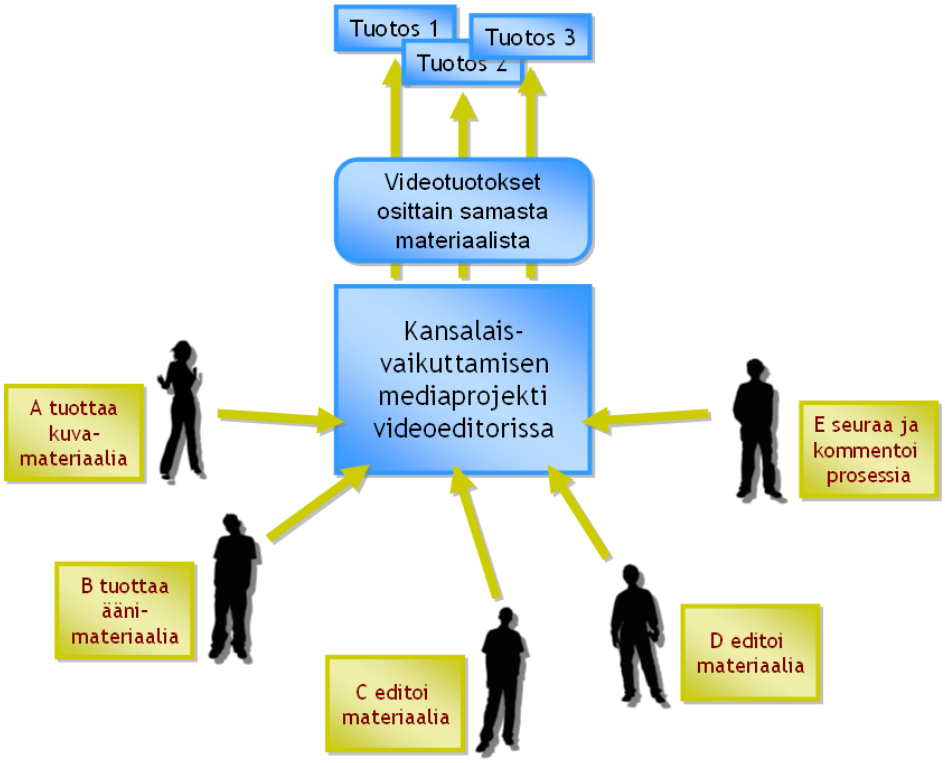
Editorin käyttöliittymään oleellisesti kuuluvien JavaScript- ja AJAX-toiminnallisuuksien toteuttaminen annettiin verkkoviestinnän toisen vuoden opiskelijoista koostuvan ryhmän hoidettavaksi. JavaScript-ryhmä toimii tiiviissä yhteistyössä palvelinryhmän kanssa toteuttaen vuorovaikutteiset käyttöliittymäkomponentit ja yhteisöllisen osallistumisen työkalut.

Foogan rinnalla on alusta asti kehitetty Django-pohjaista oppimisympäristöä, joka tulee käyttämään Foogan editorilla tuotettuja videoita osana opetusmateriaalia. Oppimisympäristön kehittämisprojekti on Foogan kehityksestä täysin irrallinen, joskin yhteistyötä palvelujen integroinnin mahdollistamiseksi tehdään.

2.1 Fooga-projektin pitkän aikavälin tavoitteet ja vaatimukset

VIDEOS-hankkeessa Foogan kehittämiselle on allokoitu kolme vuotta aikaa. Tavoitteeksi asetettiin "oikean" videoeditointisovelluksen kaltaisen www-sovelluksen toteuttaminen, mahdollisimman pitkälle samankaltaisilla

ominaisuuksilla kuin esimerkiksi ammattitason videoeditointisovellus Adobe Premiere. Kehityskaarensa aikana Foogan ominaisuuksiin on tarkoituksena lisätä yhden hengen työskentelyn rinnalle uutena ulottuvuutena usean käyttäjän samanaikainen työskentely [1]. Usean käyttäjän osallistumisella voidaan tutkia vertaistuotantoprosessia, jossa eri osanottajat toimivat projektin kannalta eri rooleissa ja tuovat oman asiantuntemuksensa prosessiin (kuva 2). Fooga pitäisi myös yllä Wikipedia-tyyppistä muutoshistoriaa projekteista, jolloin jokainen projektiin tehty muutos tallennettaisiin tietokantaan. Tarpeen vaatiessa käyttäjä voisi palata johonkin vanhaan versioon ja jatkaa työskentelyä siitä.

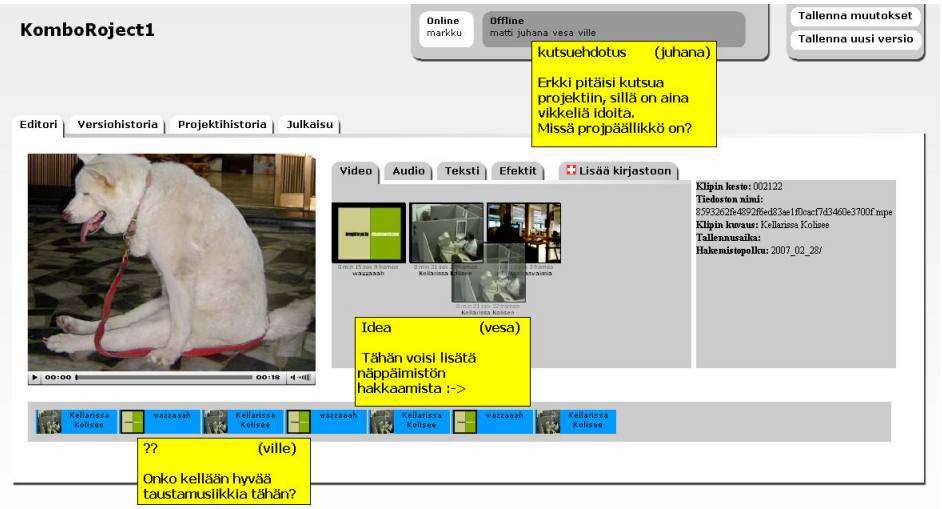


Kuva 2. Vertaistuotantoprosessi [1]

2.2 Käyttöliittymä

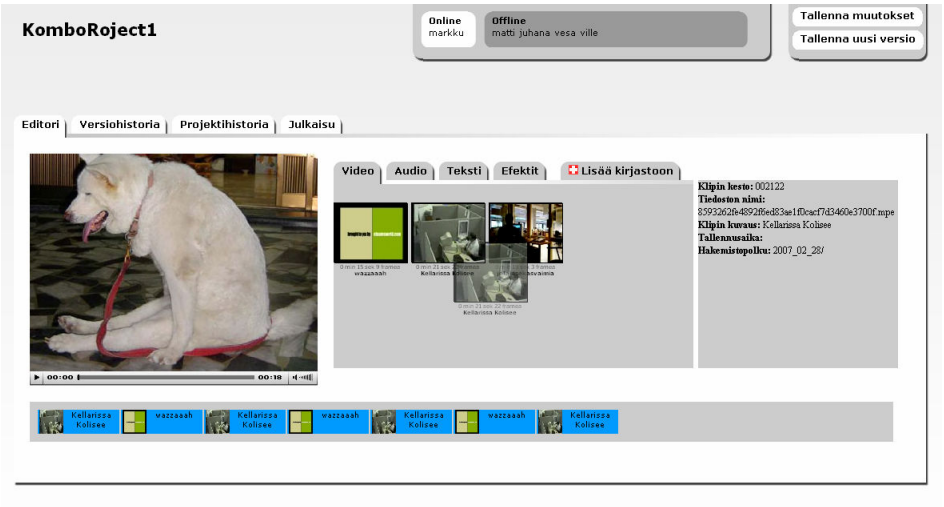
Käyttöliittymän suunnittelua varten VIDEOS-hankkeessa oli mukana kaksi käyttöliittymäsuunnittelijaa, joiden suunnitelmien ja niihin perustuvien CSS-tiedostojen (Cascading Style Sheet) perusteella Foogan käyttöliittymä toteutettiin. Koska Fooga on www-pohjainen sovellus, se perustuu asiakas-palvelinmalliin (client-server). Foogan käyttöliittymän käytännön toteutus annettiin pääosin asiakaspuolen toimintoja hoitavalle JavaScript-ryhmälle. Editorin pääidea ja muista www-pohjaisista videoeditoreista, kuten EyeSpot Betasta [2], poikkeava lähestymistapa oli dynaaminen aikajana, avoin lähdekoodi ja usean käyttäjän yhtäaikaisen samassa projektissa työskentelyn tukeminen. Dynaamisella aikajanalla jokaista videoleikettä kuvaavalla elementillä on videoleikkeen pituutta vastaava koko. Videoleikkeitä voidaan sijoittaa vapaasti aikajanalle, joka ei EyeSpotista poiketen perustu kiinteään ruudukkoon. Komponenttien raahaamiseen (drag & drop) perustuva toiminnallisuus on mukana kaikissa ammattikäyttöön tarkoitetuissa videoeditoreissa, joten sen toteuttaminen Foogassa on perusteltua.

Foogassa käyttäjien välinen kommunikointi käyttää AJAX- ja JavaScript-pohjaista keskustelujärjestelmää, jossa osa viesteistä välitetään keskustelussa (chat) ja osa viesteistä voidaan jättää muiden käyttäjien näkyville elektronisina muistilappuina. Keskustelut ja muistilaput tallennetaan forum-tyyppiseen projektikohtaiseen tietorakenteeseen, ja ne voidaan tuoda näkyviin ja poistaa näkyvistä käyttäjän valintojen mukaisesti milloin tahansa. Muistilappu voidaan asettaa ruudulle siten, että se keskittää käyttäjän huomion tiettyyn kohtaan esimerkiksi aikajanalla. Lapun viesti kertoo, mitä kyseisessä kohdassa olisi toisen käyttäjän mielestä syytä tehdä tai huomioida. Kuvassa kolme on kuvattuna muistilappujen käyttötapa ja toiminta-ajatus.



Kuva 3. Muistilappujen toiminta-ajatus

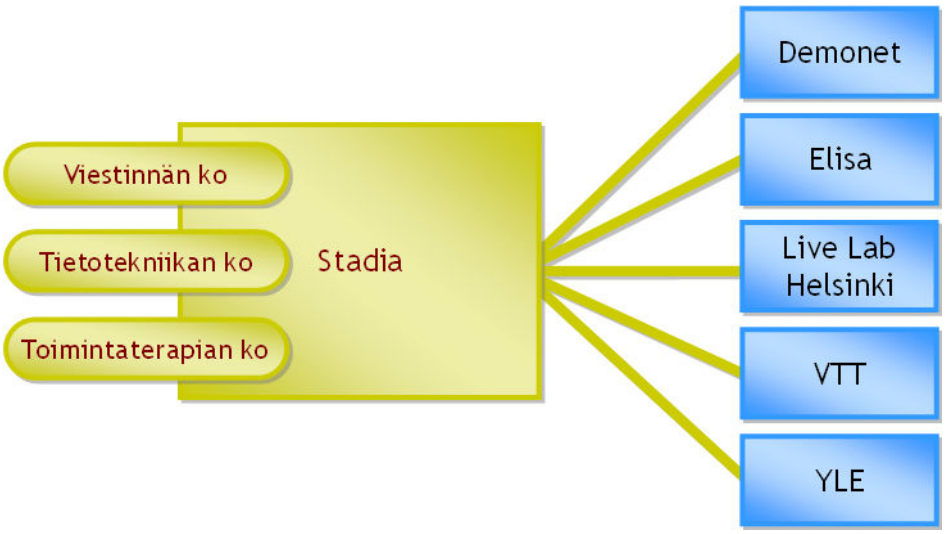
Erilaisten objektien raahaamiseen perustuvan toimintatavan ajateltiin olevan ihmiselle luontainen tapa toimia ja tietokoneisiin tottumattomallekin helppo oppia, joten päätettiin, että se tulisi olemaan Foogan käyttöliittymän perusajatus. Ylläolevassa kuvassa keskellä sijaitsee projektin kirjasto. Projektikirjasto sisältää kaikki käyttäjän projektiin lisäämät videoleikkeet ja ääniraidat. Videoiden lisääminen listalta projektikirjastoon perustuisi siihen, että käyttäjä vie videoleikettä kuvastavan objektin projektikirjastoa kuvaavaan kenttään ja vastaavasti videon lisääminen projektikirjastosta alareunassa sijaitsevalle aikajanalle toimisi samalla tavalla. Aikajanalla voisi olla useita kopioita samasta leikkeestä, jokainen omalla tavallaan leikattuna (kuva 4). Foogan ensimmäisessä versiossa aikajana sisältää projektikirjastosta käyttöön otetut videoleikkeet, joista projektin lopputulos koostetaan.Videoleikkeen poistaminen aikajanalta tapahtuisi raahaamalla kyseinen videoleike aikajanan vieressä olevaan "roskakoriin".



Kuva 4. Käyttöliittymän testiversio

2.3 Tulevat tavoitteet

Vuoden 2007 jälkimmäisellä puoliskolla tavoitteena on julkistaa Foogan versio 1.0 ja jatkaa yhteistyöprojektia Live Lab Helsingin (kuva 5) kanssa aiheella "Nuorten kansalaisvaikuttaminen". LLH-produktion demo esitetään Kallio kipinöi -tapahtumassa. Ruskeasuon koulun kanssa toteutetaan produktio ja sen tuloksia analysoidaan, tuloksia esitetään lähi-tv- tai mobiilitv-lähetyksenä. [1.]



Kuva 5. VIDEOS-hankkeen osallistujat ja yhteistyökumppanit [1]

Vuonna 2008 tavoitteena on jatkaa ohjelmiston kehitystä edelleen sekä tarjota mahdollisuuksia ohjelmistotekniikan ja viestinnän alojen lopputöille. Foogaa on tarkoitus käyttää "Kunnallisvaalit 2008" -projektissa. Yhteistyöprojektia Live Lab Helsingin kanssa jatketaan kahdella produktiolla kesän ja syksyn aikana. [1.]

Kuten kuvasta kolme nähdään, VIDEOS-hankkeella on useita yhteistyökumppaneita, joiden kanssa toteutettavissa projekteissa Foogaa aiotaan hyödyntää. Foogaa ja Opetusympäristöä on myös tarkoitus käyttää osana mediatekniikan alan opintokokonaisuuksissa kevät- ja syyslukukausilla. [1.]

Vuodelle 2009 on asetettu tavoitteeksi tarjota kahdelle viestinnän alan ja yhdelle ylemmän korkeakoulututkinnon opiskelijalle lopputyöaihe. Vuoden alussa suoritetaan projektin loppuarviointi ja tutkimustulosten analysointi. Projektin loppuraportointi ja tulosten jakaminen tapahtuvat loppuvuodesta 2009. [1.]

2.4 OHJELMISTOTUOTANTOMENETELMÄN VALINTA JA KUVAUS

Syksyllä 2006 pidetyssä aloituspalaverissa todettiin, että toteuttavan ryhmän pienestä koosta sekä alati muuttuvista vaatimusmäärittelyistä johtuen perinteiset ohjelmistotuotantomenetelmät eivät sovellu noudatettavaksi. Päätettiin, että sovelletaan ketteristä menetelmistä tuttuja toimintamalleja. Ketterissä menetelmissä pääpaino on pienillä, helposti toteutettavilla kehitysaskeleilla, jotka muodostetaan yksittäisten käyttötapausten pohjalta [3].

Etuina tällä toimintatavalla on se, että asiakas on helposti ja nopeasti tavoitettavissa palautteen antamista varten, asiakas näkee sovelluksen kehittymisen ja voi antaa välittömästi korjausehdotuksia ja palautetta. Koska kehitysaskeleet ovat pieniä ja helposti toteutettavissa, sovelluksen edistyminen näkyy selkeästi ja kehittäminen on nopeaa. Myöhäisessäkään projektin toteutusvaiheessa annettu uusi käyttötapausmääritys ei aiheuta ketteriä menetelmiä käyttävässä projektissa samanlaista alkuun palaamista

ja toteutuksen uudelleenmäärittelyä kuin esimerkiksi kankeassa vesiputousmallissa. [3.]

3 KEHITYSYMPÄRISTÖ

Foogan kehitysympäristönä käytetään kaksiydinprosessorilla varustettua työasemaa, jossa käyttöjärjestelmänä on Ubuntu Linux 6.10 (Edgy Eft). Kovalevyt on kahdennettu RAID-1-järjestelmällä, jotta kovalevyn rikkoutuminen ei aiheuttaisi kaiken tiedon menettämistä. Projektissa käytettävien tiedostojen varmuuskopiointi suoritetaan manuaalisesti kopioimalla ne pakattuna toiselle palvelimelle, joka on nauhavarmennettu.

3.1 Komentorivipohjaiset ohjelmat

Videoiden muokkaamiseen käytetään valmiita komentorivipohjaisia avoimen lähdekoodin sovelluksia, sillä nämä jo olemassa olevat sovellukset tarjoavat Foogan tarvitsemat palvelut ja omien vastaavien tekemiseen ei ollut käytettävissä resursseja. Kaikki videotiedostojen leikkaus-, yhdistämis- ja konvertointitoiminnot suoritetaan neljää eri apuohjelmaa käyttämällä.

Projektissa käytetään versionhallintaan Subversionia (SVN). Ruby on Rails tarjoaa Subversionin käyttöä helpottavia ominaisuuksia, joten SVN:n valinta oli luonteva. Kaikki Fooga-sovelluksen tiedostot ovat versionhallinnan alaisena. Subversion-palvelin asennettiin standalone-tilassa, koska Apache-web-palvelinta ei oltu vielä asennettu Foogan palvelimelle. Subversionin standalone-palvelin tarjoaa myös tietoturvaa lisäävät perusautorisointi- ja -autentikointitoiminnot [4].

3.1.1 FFmpeg

Käyttäjän lähettäessä (upload) jotakin ennaltamäärittämätöntä formaattia käyttävää videotiedoston Fooga-palvelimelle Fooga käyttää FFmpegiä

kerätäkseen transkoodausta varten metatietoa videosta. Tämän jälkeen Fooga transkoodaa vastaanotetun videon FFmpeg:llä ennaltamäärätyn kokoiseksi flash-videoksi ja tallentaa siitä levylle yhden PNG-muotoisen pienkuvan (frame), jota käytetään videon graafisena tunnisteena käyttöliittymässä.

FFmpeg:n yleisesti jaettava linux-versio ei tue suoraan esimerkiksi mp3- ja kolmannen sukupolven matkapuhelimien 3gp-videotiedostoissa käyttämiä AMR-äänikoodekkeja [5], joten Foogaa varten käännettiin versio, johon lisättiin tuki useille eri formaatille. Laaja tuki tarvitaan, jotta käyttäjien videot toimisivat palvelussa oikein. Formaattien tiukka rajaaminen helpottaisi toteutusta mutta heikentäisi loppukäyttäjän vapautta ja sovelluksen käytön mukavuutta.

Foogaan lähetetyn videon tallennuksen jälkeen FFmpeg:llä kerätään videosta kaikki sen käsittelyyn ja esittämiseen tarvittu tieto. Videoiden konvertointia varten kirjoitettu transkoodaus-skripti käynnistää FFmpeg-prosessin, joka tutkii videotiedostosta tarvittavat metatiedot suorittamatta varsinaista transkoodausta vielä tässä vaiheessa. Tulostusvirrasta kerätään talteen videon resoluutio, ääniraidan olemassaolo sekä kuvanopeus (FPS). Näiden tietojen avulla selvitetään videotiedoston kuvasuhde (aspect ratio), jonka perusteella valitaan ne parametrit, joilla FFmpeg:tä kutsutaan FLV-videon (flash-video) kääntämiseksi ja pienkuvan kaappaamiseksi. Videon transkoodauksen yhteydessä lasketaan videon kesto yksittäisinä kuvina (frame), jonka avulla lasketaan videon kesto sekunneissa ja edelleen eri aikayksiköissä. Tietojen keräämisen jälkeen kaikki tarpeellinen tieto uudesta videoleikkeestä tallennetaan tietokantaan.

```
ffmpeg -i <filename_withext> 2>&1
```

Ylläoleva käsky suorittaa FFmpeg:n pelkällä input-virralla, jolloin FFmpeg tulostaa syötteenä välitetyn videotiedoston parametrit ja antaa virheilmoituksen puuttuvasta ulossyöttövirrasta:

```
Input #0, mpeg, from 'pub-
lic/video/original/2007_03_20/18a1d358ff8228c37a17844ed7519d75
```

```
80a31e20.mpg':
Duration: 00:06:43.1, start: 0.431467, bitrate: 3039 kb/s
Stream #0.0[0x1e0]: Video: mpeg2video, yuv420p, 720x576, 7200
kb/s, 25.00 fps(r)
Stream #0.1[0x1c0]: Audio: mp2, 48000 Hz, stereo, 384 kb/s
Must supply at least one output file
```

Kaikkia aikajanalle lisättyjä videoita voidaan leikata muokkaamalla niiden alku- ja loppuaikoja. Videoleikkeiden leikkauskohdat on tallennettu tietokantaan. Kun video leikataan, FFmpeg:lle annetaan parametreinä alkuperäisen videotiedoston sijainti, käyttöliittymässä määritellyt leikkeen alkuhetki ja kokonaiskesto, kohdeformaatti sekä videon bitrate. Tuloksena FFmpeg luo juuri halutun mittaisen videotiedoston valitussa formaatissa. Alkuperäistä videotiedostoa ei koskaan muuteta tai tuhota.

Seuraavanlaisella käskyllä alkuperäisestä videosta luodaan FLV-video ennaltamäärätyillä bittimäärillä ja alkuperäisen videon kuvasuhteen mukaan määrittyvällä resoluutiolla [6]. Ääniraidan laatu ja näytteenottotaajuus sekä äänikanavien määrä pakotetaan myös ennaltamäärättyyn muotoon:

```
ffmpeg -i <filename_withext> -b 360 -r 25 -s <resolu-
tion_width>x<resolution_height> -ab 96 -ar 22050 -ac 1 <file-
name_noext>.flv 2>&1
```

```
Ylläolevassa käskyssä parametrit tarkoittavat seuraavaa: -i
määrittää syöttövirran eli käsiteltävän videotiedoston, -b
määrittää videovirran bittinopeuden (bitrate), -r määrittää
videon kuvataajuuden (fps) ja -s asettaa resoluution. Vipu -ab
asettaa äänivirran bittinopeuden, -ar äänivirran
näytteenottotaajuuden ja ac-vipu asettaa äänikanavien määrän.
```

3.1.2 Mencoder

Mencoder on MPlayer-projektin ohjelma, jolla voidaan muuntaa videotiedostoja toiseen formaattiin, muokata ja yhdistää niitä yhdeksi

videotiedostoksi. FFmpeg:stä puuttuu tuki useiden videoiden yhdistämiselle, joten Mencoderin käyttäminen on Fooga-projektissa välttämätöntä. FFmpeg ja Mencoder käyttävät samoja videokoodekkikirjastoja, niiden välillä ei pitäisi ilmetä yhteensopivuusongelmia. Lisäksi Mencoder tarjoaa erilaisia suodattimia (filter) videoiden muokkaamiseen, skaalaamiseen ja rajaamiseen (crop). Näitä ominaisuuksia voidaan hyödyntää Foogan myöhemmissä kehitysversioissa. [7.]

Allaoleva esimerkki näyttää, miten Mencoderia käytetään Foogassa videoleikkeiden yhdistämiseen:

```
mencoder -forceidx -oac copy -ovc copy -noskip -o <file-
name_noext>.avi <videos> 1>/dev/null 2>&1
```

Forceidx-parametri pakottaa Mencoderin uudelleenkirjoittamaan keyframet uudelleen, -oac määrittää ulostulovirran äänikoodekin (output audio codec), -ovc-parametri määrittää ulostulovirran videokoodekin (output video codec). Parametrilla -o määritellään lopputuloksena syntyvä videotiedosto ja sen jälkeen luetellaan kaikki yhteen liitettävät videotiedostot.

3.1.3 FLVTool2

FLVTool2 on Ruby-ohjelmointikielellä ohjelmoitu sovellus, jolla voidaan korjata FLV-tiedoston virheellinen metadata [8]. FFmpeg:llä konvertoidut FLV-tiedostot videot ovat metadataltaan virheellisiä ja Foogassa FLVTool2:n avulla nämä tiedot korjataan. Mikäli metatietokenttiä ei korjata, FLV-videota selaimessa toistettaessa aikajanalla ei voida kelata videota toiseen kohtaan, ja videon kesto näkyy virheellisenä.

3.1.4 ImageMagick

ImageMagick on monipuolinen komentorivipohjainen sovellus, jolla voi helposti muokata kuvaa. Esimerkiksi konvertointi ja kuvakoon muuttaminen ovat ImageMagickin perustoimintoja. Foogan ensimmäisessä versiossa ImageMagickia käytetään videoista otettujen pienkuvien (thumbnail) konvertoimiseen PNG-kuvaformaatista JPG-kuvaformaattiin.

Projektin edetessä siirryttäneen käyttämään ImageMagickin kirjastojen ja Rubyn väliin sijoittuvaa rajapintaa RMagick:iä. RMagickiä käytettäessä kuvankäsittelykäskyjä voidaan suorittaa suoraan Ruby-koodista ilman, että joudutaan suorittamaan komentorivikäskyjä [9].

3.2 MySQL

Fooga-projektissa tiedonhallintajärjestelmänä käytetään MySQL-relaatiokantaa [10]. Valintaa vaikuttivat aikaisempi kokemus MySQL:stä sekä työkalujen laaja saatavuus ja helppokäyttöisyys. Ruby on Rails tukee laajaa joukkoa tietokantoja ja tietokannan vaihtaminen on vaivatonta muokkaamalla config/-hakemistossa olevaa database.yml-tiedostosta muutamaa riviä.

3.3 Ruby

Ruby on Yukihiro "matz" Matsumoton kehittelemä olio-ohjelmointikieli. Matsumoto on yhdistänyt Rubyyn lempiohjelmointikieltensä Perlin, Smalltalkin, Eiffelin, Adan ja Lispin piirteitä. Tarkoituksena oli luoda tehokas oliokeskeinen skriptikieli. Matsumoto onnistui tavoitteissaan niin hyvin, että kaikki Ruby:ssa ovat oliota. Esimerkkikoodi demonstroi, kuinka tätä piirrettä voidaan hyödyntää. [11.]

```
10.times do
  puts "Hello World!"
end
```

Ruby on täysin ilmainen ja laajennettavissa oleva ohjelmointikieli. Ruby-luokkakirjasto on täysin muokattavissa työn alla olevasta ohjelmasta käsin, joten esimerkiksi valmiisiin luokkiin voidaan lisätä uusia metodeja ilman, että luokkaa tarvitsee periä [11]. Allaolevassa esimerkissä on havainnollistettu, miten Numeric-kantaluokkaan voidaan lisätä plus-metodi, joka toimii +-metodin rinnalla:

```
class Numeric
  def plus(x)
    self.+(x)
  end
end

y = 5.plus 6
# y is now equal to 11
```

Aloittelevan Ruby-ohjelmoijan on hyvä tutustua ensimmäisenä irb-nimiseen ohjelmaan, jolla Ruby-koodia voidaan kirjoittaa ja ajaa täysin interaktiivisesti konsolista käsin. Ohessa on esimerkki irb-istunnosta (koodista on poistettu muutama ylimääräinen välitulostus, jonka irb antaa).

```
irb(main):001:0> viikonpaivat = %w[maanantai tiistai
keskiviikko torstai perjantai lauantai sunnuntai]

irb(main):002:0> viikonpaivat.each do
irb(main):003:1*   |paiva|
irb(main):004:1*   puts paiva.capitalize
irb(main):005:1> end
```

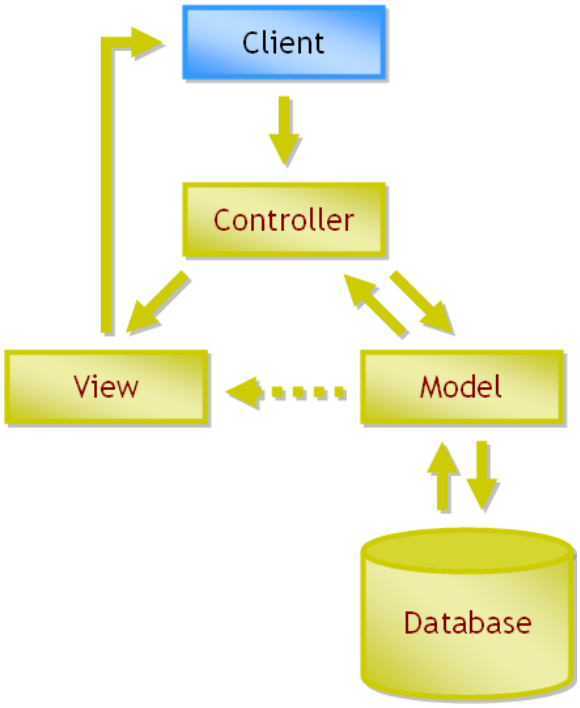
```
Maanantai
Tiistai
Keskiviikko
Torstai
Perjantai
Lauantai
Sunnuntai

irb(main):006:0> viikonpaivat.reverse.each do
irb(main):007:1*   |paiva|
irb(main):008:1*   puts paiva.upcase
irb(main):009:1> end


SUNNUNTAI
LAUANTAI
PERJANTAI
TORSTAI
KESKIVIIKKO
TIISTAI
MAANANTAI
```

3.4 Ruby on Rails

Ruby on Rails on nuoren tanskalaisen David Heinemeier Hanssonin vuonna 2004 julkaisema web-sovelluskehys. Ruby on Rails syntyi sivutuotteena kun David Heinermeier Hansson kehitti Basecamp-nimistä WWW-pohjaista projektityöskentelysovellusta [12]. Ruby on Rails pohjautuu MVC-arkkitehtuuriin, jossa näkymät, sovelluslogiikka ja ohjelman talletettu tila ja tieto ovat erotettu toisistaan. Railsin MVC-suunnittelumallissa Controller vastaanottaa HTTP-pyyntöjä asiakkaalta (WWW-selain) ja pyytää Modelia hakemaan tarvittava tieto tietokannasta ja siirtää tiedon esitettäväksi View:lle, joka on HTML-sivu (kuva 6).



Kuva 6. Ruby on Rails-sovelluksen rakenne [13, s. 10]

Ruby on Railsin API:n keskeisimmät komponentit ovat

- ActionController
- ActionMailer
- ActionView
- ActionWebService
- ActiveRecord
- ActiveSupport.

ActionController

ActionController on WWW-pohjaisten kutsujen ydin Ruby on Rails-sovelluskehyksessä ja se edustaa MVC-arkkitehtuurin kontrolleria. ActionController-luokka sisältää joukon metodeja (actions), joita voidaan kutsua WWW-kutsun yhteydessä. Action-metodit tyypillisesti tulostavat WWW-sivupohjan (template) tai ohjaavat kontrollin toiselle actionille. Useimmissa action-metodeissa tietokannasta haetaan joko olio tai

taulukollinen olioita, jotka asetetaan jäsenmuuttujiksi, jotta tarvittavat tiedot voidaan esittään varsinaisessa WWW-näkymässä. Esimerkkikoodissa kontrolleri sisältää kaksi action-metodia, joilla voidaan listata käyttäjiä ja poistaa yksittäinen käyttäjä.

```
class UserController < ApplicationController

  def list
    @users = User.find(:all)
  end

  def delete
    User.find(params[:id]).destroy
  end

end
```

ActionMailer

ActionMailer-luokka tarjoaa helpon tavan lähettää sähköpostiviestejä Railsista. ActionMailer-luokalla voidaan ohjelmoida käteviä sähköpostirobootteja, jotka osaavat muotoilla viestit oikeanlaisiksi ja lähettää vastaanottajille. Oheinen esimerkkikoodi [14, ActionMailer::Base] demonstroi, kuinka käyttäjätilin rekisteröinnin yhteydessä voidaan lähettää tietoa rekisteröitymisen onnistumisesta sähköpostitse käyttäjälle.

```
class Notifier < ActionMailer::Base
  def signup_notification(recipient)
    recipients recipient.email_address_with_name
    from       "system@example.com"
    subject    "New account information"
    body       :account => recipient
  end
end
```

ActionView

ActionView-luokka edustaa WWW-näkymää (MVC-arkkihtehtuurin View) eli varsinaista HTML-sivua. HTML-sivut ovat Ruby on Railsissa sivupohjia joihin on upotettu Ruby-koodia. Käytettäviä sivupohjatekniikoita on ERb (.rhtml), RXML (.rxml) ja RJS (.rjs). Tyypillisesti käytetään ERb-tekniikkaa, johon Ruby koodia upotetaan käyttämällä <% %>-notaatiota. ActionController-luokassa jäsenmuuttujiksi asetetut oliot ovat käytettävissä ActionView-sivupohjissa, Niiden pohjilta on helppoa tulostaa dynaamisia sivuja. Esimerkkikoodissa tulostetaan jokaisen User-olion tiedot, joka on @users-taulukossa. [14, ActionView::Base.]

```
<% for user in @users %>
  <p>Username: <%= user.username %></p>
  <p>First name: <%= user.firstname %></p>
  <p>Last name: <%= user.lastname %></p>
  <p>E-mail: <%= user.email %></p>
<% end %>
```

ActionWebService

ActionWebService on Ruby on Railsin tuoreimpia ominaisuuksia. Se tarjoaa tyypilliset web-palveluprotokollat käytettäväksi. Näihin kuuluvat mm. SOAP, XML-RPC ja dynaaminen WSDL-määrityksen luonti [15] Railsin tarjotessa yleiset web-rajapinnat, voidaan Railsia hyödyntää vaivattomasti esimerkiksi .NET-sovellusten kanssa kommunikointiin. Esimerkkikoodissa [14, ActionWebService::Base] esitellään, kuinka web-palvelu ohjelmoidaan ja kuinka määritellään, mitä web-palvelu ottaa asiakkaalta vastaan ja mitä se palauttaa asiakkaalle.

```
class PersonService < ActionWebService::Base
  web_service_api PersonAPI

  def find_person(criteria)
    Person.find(:all) [...]
  end

  def delete_person(id)
    Person.find_by_id(id).destroy
  end

end

class PersonAPI < ActionWebService::API::Base
  api_method :find_person, :expects => [SearchCriteria],
:returns => [[Person]]
  api_method :delete_person, :expects => [:int]
end

class SearchCriteria < ActionWebService::Struct
  member :firstname, :string
  member :lastname,  :string
  member :email,     :string
end
```

ActiveRecord

ActiveRecord on yksi Ruby on Rails -sovelluskehyksen tärkeimmistä komponenteista, koska se mahdollistaa vaivattoman yhteyden tietokantaan ja olioiden tallentamisen ja hakemisen kannasta [14, ActiveRecord::Base]. ActiveRecord noudattaa samannimisen Object-Relational Mapping tekniikan filosofiaa. ActiveRecord edustaa MVC-arkkitehtuurin Model-osaa pitämällä huolta tiedon säilyttämisesta.

ActiveSupport

ActiveSupport on luokkakirjasto, joka tarjoaa apuluokkia ja metodeita, jotka on havaittu hyödyllisiksi Ruby on Rails sovelluskehyksessä [14, ActiveSupport]. Allaoleva esimerkkikoodi havainnollistaa kuinka ActiveSupport-kirjaston apufunktiot yksinkertaistavat koodia ja tekevät siitä samalla luettavampaa. Koodissa lasketaan istuntotaulukkoon ajanhetki, jolloin istunto vanhenee ellei käyttäjä kyseisen aikarajan sisällä lataa sivua.

```
session[:expiry_time] = 30.minutes.from_now
```

Käytännöt

Ruby on Rails pyrkii myös olemaan helppo käyttää ja pyrkimys on myös, että sen avulla olisi helppo kehittää web-sovelluksia. Tähän on pyritty vähentämällä konfigurointitarvetta ja korvaamalla konfigurointia ennaltamäärätyillä ohjelmointikäytännöillä [16]. Esimerkiksi noudattamalla tiettyjä nimeämiskäytäntöjä ohjelmoija voi säästyä suurelta määrältä konfigurointeja. Yksi tärkeimmistä käytännöistä on sovelluksen hakemistorakenne, joka on oletuksena seuraavanlainen:

```
app
  controllers
  helpers
  models
  views
    layouts
components
config
db
doc
lib
log
public
script
```

```
test
unit
tmp
vendor
```

App-hakemistossa sijaitsevat Rails-sovelluksen tärkeimmät tiedostot. Hakemisto on jaettu MVC-mallin mukaisesti kontrollereihin, vieweihin ja modelehin. Neljäntenä app-hakemistossa on helper-luokat, jotka tarjoavat jokaiselle viewille ohjelmoijan määrittelemiä apumetodeja, jotta view-sivupohja ei tulisi liian monimutkaiseksi. Hakemistossa script sijaitsee apuohjelmia, joilla MVC-mallin komponentteja voidaan helposti luoda. Näitä kutsutaan generaattoreiksi Rails-ympäristössä. Generoimalla esimerkiksi modelin luo Rails samalla testaukseen tarvittavat tiedostot oikeisiin hakemistoihin.

Routes

Routes on tehokas moduuli Ruby on Rails -sovelluskehyksessä, jolla voidaan koodata web-sivuston osoitteet käyttäjä- ja hakukoneystävälliseen muotoon. Toiminnaltaan se on hyvin samankaltainen kuin Apachen mod_rewrite-moduuli, mutta huomattavasti helppokäyttöisempi. Rails-sovelluksessa config/ hakemistossa on routes.rb tiedosto, jossa kunkin Rails-sovelluksen URL-reititys on määritelty. Vakiona tämän tiedoston sisältö on seuraavanlainen (koodista on poistettu kommentoidut ohjeet). [17.]

```
ActionController::Routing::Routes.draw do |map|
  map.connect ':controller/service.wsdl', :action => 'wsdl'
  map.connect ':controller/:action/:id.:format'
  map.connect ':controller/:action/:id'
end
```

Tiedoston sisällöstä voidaan todeta, että ensimmäinen map.connect-käsky määrittää reitityksen web-palvelun kuvaukselle. Toinen käsky määrittää yleisen kontrolleri/toiminto/parametri-muodon siten, että parametrin lisäksi voi asettaa haluamansa formaatin, joka mahdollistaa esimerkiksi .html-päätteiset sivut. Alimmaisena oleva reititys on formaatin asetusta lukuunottamatta sama kuin edellinenkin ja toimii oletuksena. Eli jos Rails saa HTTP-pyynnön osoitteeseen "user/delete/1", tulkitsee Rails tämän pyynnön kohdistuvan kontrollerille user, action-metodille delete parametrilla numeroarvo 1. Tähän routes.rb-tiedostoon voidaan määrittää itse haluamiaan reitityksiä. Seuraava koodiesimerkki on reititys Foogan eräästä toiminnosta:

```
map.connect 'projects/add_video_to_project_time-
line/:project_id/:video_id', :controller => 'projects',
:action => 'add_video_to_project_timeline'
```

Tässä reitityksessä tarkoituksena oli välittää kaksi numeerista parametria kutsuttavalle action-metodille. Oletusreitityksellä pystyi välittämään vain yhden numeerisen parametrin, joten tässä tapauksessa oli tarpeen saada välitettyä toinenkin. Kutsuttava URL-osoite on nyt esimerkiksi "`http://domain.tld/projects/add_video_to_project_timeline/3/8`".

4 OBJECT-RELATIONAL MAPPING

Taulupohjaiset tietokannat on kehitetty ennen varsinaisen oliopohjaisen ohjelmoinnin syntyä. Tietoja tallennettiin taulurakenteisiin ja ajan myötä relaatiokantojen käyttö tuli ihmisille tutuksi ja levisi laajaan käyttöön. Kun lopulta oliopohjainen ohjelmointi yleistyi, huomattiin, että oliopohjaisen sovelluksen ja sitä tukevan relaatiotietokannan välille syntyi tietynlainen kuilu. Tiedon hakeminen taulurakenteista ja niiden muuttaminen olioiksi oli turhan hankalaa ja työlästä. [18; 19]

Syntyi tarve tekniikalle, jolla tämä kuilu saataisiin kavennettua tai poistettua täysin. Object-Relational Mapping (ORM) pyrkii vastaamaan haasteeseen, kuinka tietokannan tauluissa olevat rivit pystytään muuttamaan olioiksi ja takaisin, jotta tiedon käsittely sovelluksessa olisi helppoa eikä vaadittaisi valtavaa määrää koodirivejä hakemaan tietoa tietokannasta. Tyypillisesti tieto haetaan relaatiokannoista käyttämällä SQL-kieltä (Structured Query Language). ORM:n tarkoitus on piilottaa ohjelmoijalta tarvittavat yhteyksien muodostukset ja SQL-lauseet, joilla tietoa haetaan. [18; 19]

Tuomalla ylimääräinen kerros tietokannan ja sovelluksen välille tarjotaan myös muita merkittäviä etuja, kuten tietokantariippumattomuus, tiedon validointi ja tietoturva sekä transaktioiden hallinta ja tiedon eheys. Tietokanta-riippumattomuus mahdollistaa käytettävän tietokannan piilottamisen varsinaiselta sovellukselta sallimalla täten monipuoliset yhteydet eri tietokantatyypeille. Tiedon validointi ja tietoturva tarkoittaa tässä tapauksessa sitä, että ylimääräinen ORM-kerros pystyy huolehtimaan, että tietokantaan ei virheellistä, turvatonta tai puutteellista tietoa pystytä tallentamaan. [18; 19]

4.1 Active Record

Active Record suunnittelumalli on yksi ORM-tekniikka, jolla pyritään helpottamaan tietojen hakua ja tallennusta tietokantaan. Active Record on olennainen osa Ruby on Rails –sovelluskehystä. Täten tarjotaan kehittyneitä ORM-ominaisuuksia Ruby on Railsin käyttäjille. Active Record suunnittelumallin esitti Martin Fowler -niminen tunnettu oliopohjaisen ohjelmoinnin puolestapuhuja kirjassaan "Patterns of Enterprise Application Architecture" [20]. Active Record toteutuksen ohjelmoi Ruby on Rails -sovelluskehykseen tanskalainen David Heinemeier Hansson, joka on Ruby on Railsin pääkehittäjä.

Active Record -suunnittelumallissa luokka edustaa taulua, luokan ilmentymä edustaa taulun riviä ja luokan ilmentymän jäsenmuuttujat edustavat taulun rivin sarakkeiden arvoja (ks. kuva 7). Taulujen yhdistämisestä on myös mahdollista saada olioita. Active Record -luokka siis kapseloi tietokantayhteyden, käärii tietokannan rivin ilmentymäänsä ja tarjoaa mahdollisuuden lisätä business-logiikan toimintoja ilmentymäänsä. [20]

users				
id	username	firstname	lastname	email
1	john	John	Doe	john@doe.com
2	jane	Jane	Doe	jane@doe.com
3	jake	Jake	Doe	jake@doe.com
4	jill	Jill	Doe	jill@doe.com
.	.	.	.	.
.	.	.	.	.

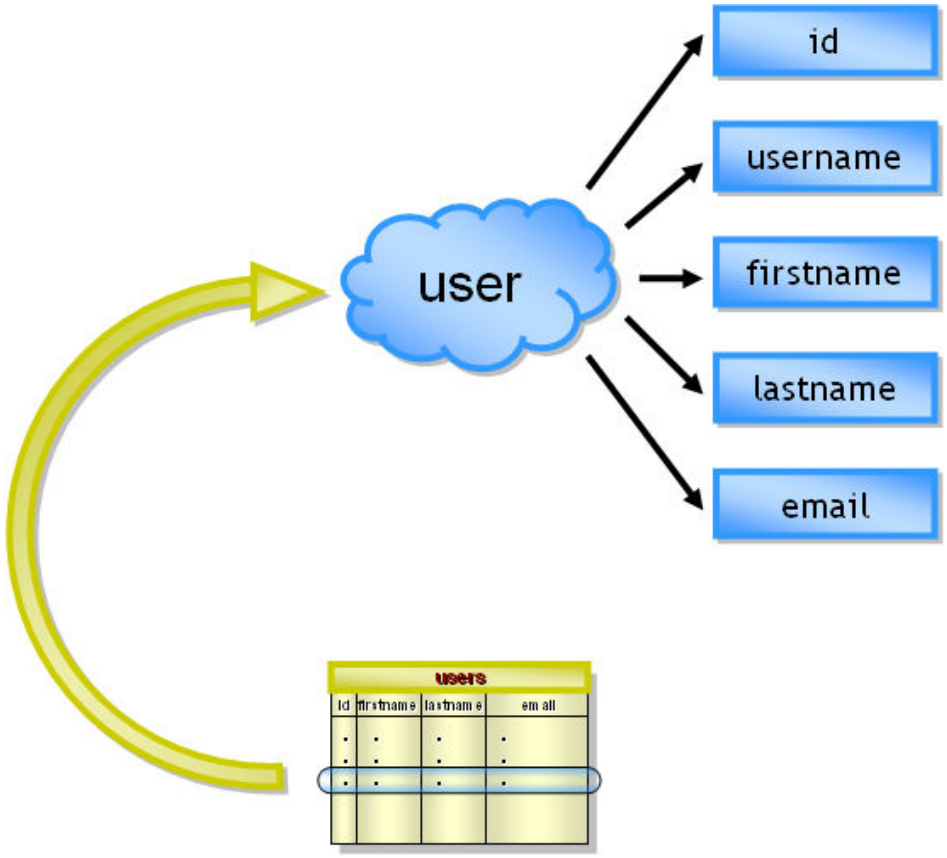
Kuva 7. Users-taulu

Ruby on Rails -nimeämiskäytäntöjen avulla ORM-yhteys tietokannan tauluihin käy todella helposti luomalla sopivasti nimettyjä model-luokkia. Esimerkiksi jos tietokannassa on taulu nimeltään *users*, on ohjelmoijan luotava User-niminen model, joka periytetään ActiveRecord-kantaluokasta. Nyt Rails osaa etsiä tietokannasta käyttäjiä users-taulusta. Itse model-luokkaan ei oletuksena tarvitse kirjoittaa riviäkään koodia. Siitä huolimatta Ruby on Railsin Active Record löytää tarvittavat tiedot itse kuten taulun sarakkeiden nimet hakemalla ne tietokannasta.

Seuraava koodiesimerkki havainnoi, kuinka Ruby on Rails -ympäristössä tietokannasta etsitään käyttäjää käyttäjänimen mukaan ja kuinka helposti tietokannan taulusta saadaan haettua olio. Kun instanssi ollaan saatu Active Recordilta, niin sähköpostiosoite vaihdetaan toiseksi ja tieto tallennetaan takaisin tietokantaan. Jos Rails ei löydä metodia, joka on muotoa find_by_xxxx, niin Rails osaa päätellä, että nyt yritetään hakea taulusta riviä sarakkeen nimen perusteella.

```
user = User.find_by_username('john')
user.email = 'john2@doe.com'
user.save
```

Muita Active Record –luokan tarjoamia metodeja on useita. Ne löytyvät helpoiten Ruby on Rails API -dokumentaatiosta [13, ActiveRecord::Base]. Käytetyin kuitenkin näistä on *find()*. ORM-muunnosta havainnolistamaan pyrkivä kuva (ks. kuva 8) näyttää, kuinka tietokannan rivi vastaa oliota, jolla on jäsenmuuttujinaan taulun sarakkeiden arvot.



Kuva 8. ORM-prosessi

Seuraava koodiesimerkki demonstroi, kuinka uusi Active Record -luokan ilmentymä luodaan ja kuinka se tallentuu tietokantaan. Active Record -luokan ja ilmentymien käyttö on täysin samanlaista kuin minkä tahansa muun Ruby-olion tai luokan käyttö. Ainoa ero tässä tapauksessa on Active Record -luokan save-metodikutsu, joka tallettaa uuden rivin tietokantaan.

```
user = User.new
user.username = 'jane'
user.firstname = 'Jane'
user.lastname = 'Doe'
user.email = 'jane@doe.com'
user.save
```

Active Record on Ruby on Railsin myötä saanut myös huomiota muissa ohjelmointipiireissä. Active Recordille löytyy toteutus esimerkiksi PHP-kielen sovelluskehyksissä CakePHP ja PHP on Trax.

Oliosuhteet ja liitostaulut

Ruby on Railsin Active Recordien välisiä suhteita kuvataan kirjoittamalla model-tiedostoihin tarvittavat suhteet. Näiden avainsanojen avulla Active Record osaa etsiä tarvittavat muut taulut ja tehdä tarvittavat liitosoperaatiot datan kokoamiseksi. Tietokantasuunnittelussa käytettävä ER-mallinnus ja siitä johdettu relaatiomalli liittyy hyvin läheisesti Railsin tapaan ilmoittaa sidoksistaan muihin luokkiin eli tauluihin. Lukumääräsuhteita kuvataan seuraavilla avainsanoilla [12, s. 333]:

- has_many
- has_one
- has_and_belongs_to_many
- belongs_to.

Kyseiset avainsanat lisäävät kyseessäolevalle model-luokalle metodeita, joilla tiedon haku onnistuu vaivattomasti linkitetyistä olioista. Seuraava koodiesimerkki pyrkii selkeyttämään olioiden välisiä suhteita Railsissa ja tietokannan tauluissa.

```
class User < ActiveRecord::Base
  has_many :posts
end
```

```
class Post < ActiveRecord::Base
  belongs_to :user
end
```

Taulukko 1. Käyttäjätaulu

users	
id	username
1	john
2	jane
3	jake
4	jill

Taulukko 2. Posts-taulu

posts				
id	title	body	created_at	user_id
1	Hello	Hello World!	######	1
2	Foo	Foobar!	######	3

Koodissa esitetyistä modeleista voidaan huomata, että käyttäjällä voi olla monta postia, mutta yhdellä postilla voi olla vain yksi käyttäjä. Tarkastelemalla tauluja (ks. taulukko 1 ja taulukko 2) voidaan todeta, että taulut tukevat tässä tapauksessa Railsin modelien määrityksiä. Seuraavat koodiesimerkit havainnollistaa kuinka, nämä olioiden väliset liitokset toimivat.

```
user = User.find(:first)
user.posts.each do |post|
  puts post.title
end

post = Post.new
post.title = ‘Test’
```

```
post.body = ‘Testing…’

user.posts << post

user.save
```

Kuten esimerkistä voidaan todeta, on kahden taulun välisten liitosten käsittely vaivatonta ja uusien rivien lisääminen helppoa.

4.2 Tietokantayhteydet Javassa

Javan tietokantayhteydet saadaan käyttämällä DriverManager-, Connection-, Statement- ja ResultSet-luokkia. Ensimmäisenä ladataan käytettävä ajuri Class.forName-metodikutsulla. Seuraavaksi yhteys tietokantaan luodaan pyytämällä Connection-luokan instanssia DriverManagerilta käyttämällä getConnection-metodia. Parametrina välitetään merkkijono, jonka perusteella DriverManager-yrittää muodostaa yhteyden merkkijonon määrittämään tietokantaan. Tämän jälkeen Connection-luokan ilmentymältä voidaan pyytää palauttamaan Statement-luokan ilmentymä, jonka avulla voidaan SQL-kyselyitä suorittaa kyseessä olevalle tietokannalle. Kyselyt suoritetaan käyttämällä executeQuery-metodia, jossa parametrina on merkkijonona SQL-kysely. Tämä metodi palauttaa mm. ResultSet-luokan ilmentymän, joka sisältää haetun datan. Tarvittavat tiedot voidaan sitten selvittää saadusta ResultSetistä. Oheinen koodiesimerkki demonstroi tietokantayhteyden muodostamista ja tiedon hakuprosessia.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection myCon = DriverManager.getConnection("jdbc:odbc:FMS");
Statement myStatement = myCon.createStatement();

String query = "SELECT * FROM TAuthors WHERE ";
```

```
query += "a_nickname LIKE '%" + hakuSana
query += "%' OR a_fname LIKE '%" + hakuSana
query += "%' OR a_lname LIKE '%" + hakuSana + "%'";

ResultSet myRs = myStatement.executeQuery(query);

while (myRs.next()) {
        String firstname = myRs.getString("a_fname");
        String lastname = myRs.getString("a_lname");
        String nickname = myRs.getString("a_nickname");
        AuthorBean ab = new AuthorBean();
        ab.setFirstname(firstname);
        ab.setLastname(lastname);
        ab.setNickname(nickname);
        list.add(ab);
}

myRs.close();
myStatement.close();
myCon.close();
```

4.2.1 DAO/DTO-suunnittelumalli

Javassa perinteinen tapa muodostaa tietokannan riveista olioita on käyttää DAO/DTO-suunnittelumallia. DAO (Data Access Object) piilottaa tietokanta-yhteyden taakseen, jolloin ohjelmoijan ei tarvitse huolehtia taustalla tapahtuvasta tietoliikenteestä vaan pyytää esimerkiksi DAO:lta olioita tietokannasta. DAO-kerroksen läpi siirrettäviä oloita kutsutaan DTO:ksi (Data Transfer Object) tai VO:ksi (Value Object). JavaBean-tekniikkaa käyttämällä DTO-olioista tehdään yksinkertaisia tiedon kapselointi-olioita, joiden jäsenmuuttujiin voidaan tehdä muutoksia käyttämällä set- tai get-alkuisia metodikutsuja. Hankaluutena DAO/DTO-mallissa on se, että ohjelmoijan on itse kirjotettava toteutus DAO- ja DTO-luokille. DTO-luokkien toteutukset ovat tyypillisesti varsin yksinkertaisia, mutta DAO-luokka on varsin

monimutkainen. DAO-luokassa on luotava yhteys tietokantaan ja huolehdittava, että käytetään tiedon siirtoon oikeita SQL-lauseita ja, että tieto kapseloidaan DTO-olioihin. Seuraavassa koodilistauksessa on esimerkki DTO-oliosta (UserBean). [21]

```
public class UserBean implements Serializable {
  private String username;
  private String firstname;
  private String lastname;
  private String email;

  public void setUsername(String username) {
    this.username = username;
  }

  public String getUsername() {
    return username;

  }

  public void setFirstName(String firstname) {
    this.firstname = firstname;
  }

  public String getFirstName() {
    return firstname;
  }

  ...
}
```

DAO/DTO-malli ei ole varsinainen ORM-tekniikka, vaan perinteinen tapa hakea tietoa kannasta ja muuttaa se olioksi. DAO/DTO-mallin päätarkoitus on yksinkertaistaa pääohjelman koodia piilottamalla varsinainen tietokantayhteys näkyvistä. Halutessaan on ohjelmoija turvauduttuva

vaihtoehtoiseen tekniikkaan, jos tarkoituksen on nimenomaisesti hyödyntää ORM-tekniikkaa. [21]

4.2.2 Hibernate

Hibernate on täydellinen avoimen lähdekoodin ORM-toteutus Java-kielelle. Se on osa JBoss:n Java Enterprise Edition ohjelmistopakettia. JBoss on nykyisin Red Hatin omistuksessa. Hibernate on laajasti käytetty ORM-ratkaisu Java-sovellusympäristössä.

Myös Hibernatessa tietoa edustaa JavaBean-luokka, joka on samanlainen kuin DTO-luokka. Oleellinen ero DAO/DTO-malliin on se, että varsinaista tiedon hakemista ja käsittelemistä tarvittavaa koodia ei tarvita. Hibernatelle on kerrottava, mille luokalle ORM-muunnos tehdään ja mitä taulua se vastaa tietokannassa. Nämä määritykset tehdään XML-konfigurointitiedostoon ja siihen on varsin yksiselitteisen tarkasti määriteltävä tallennettava luokka ja siihen kuuluvat jäsenmuuttujat sekä vastaavat sarakkeiden nimet tietokannassa. Sarakkeiden nimiä tietokannassa ei tarvitse määritellä, jos ne eivät poikkea luoka jäsenmuuttujien nimistä, mutta jäsenmuuttujat ovat siitä luolimatta listattava tähän. Jos muutoksia tehdään käytettyyn papuluokkaan, on myös tätä konfigurointitiedostoa muokattava vastaamaan muutoksia. [22]

Seuraavassa koodiesimerkissä on erästä event-luokkaa varten konfiguroitu Hibernaten XML-konfiguraatiotiedosto (Event.hbm.xml). Siinä määritetään taulun sarakkeiden nimiä vastaavat event-olion jäsenmuuttujien nimet ja tyypit Myös taulun nimi ja pääavain määritetään.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
```

```
    <class name="events.Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="native"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
    </class>

</hibernate-mapping>
```

Seuraavassa koodiesimerkissä on Java-sovellukseen tarvittava XML-konfigurointitiedosto, jonka avulla Hibernate muodostaa tietokantayhteyksiä [22]. On myös syytä huomioida, että myös tässä tiedostossa määritetään ORM-muunnosta kaipaavien luokkien Hibernate XML-konfiguraatiotiedostot. Uuden ORM-muunnosta käyttävän luokan lisääminen vaatii myös tämän projektikohtaisen konfigurointitiedoston muuttamisen varsinaisen luokkaa vastaavan konfigurointitiedoston lisäksi. Viittaukset luokkien konfigurointitiedostoon annetaan *<mapping resource=""/>*-elementin avulla. Muilla parametreilla määritellään tietokantaspesifiset tiedot kuten käyttäjänimi ja salasana.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>

    <!-- Database connection settings -->
    <property
name="connection.driver_class">org.hsqldb.jdbcDriver</property>
    <property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
```

```
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>

    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">1</property>

    <!-- SQL dialect -->
    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

    <!-- Enable Hibernate's automatic session context management -->
    <property name="current_session_context_class">thread</property>

    <!-- Disable the second-level cache  -->
    <property name="cache.provider_class">
      org.hibernate.cache.NoCacheProvider
    </property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">true</property>

    <!-- Drop and re-create the database schema on startup -->
    <property name="hbm2ddl.auto">create</property>

    <mapping resource="events/Event.hbm.xml"/>

  </session-factory>

</hibernate-configuration>
```

Seuraava ohjelmanpätkä on esimerkki, kuinka sitten Hibernatea käytetään ohjelmasta käsin [22]. Istuntotehtaalta voidaan pyytää istuntoja, jonka avulla voi tietokannasta hakea ja tallentaa olioita. Transaktioiden käsittely on myös mahdollista session-oliolla. Java-papuun kääritään tarvittavat tiedot ennen tallennusta ja sen jälkeen annetaan session-oliolle se tallennettavaksi, jolloin

Hibernate muuttaa annetun pavun SQL-kielisiksi kyselyiksi ja tallentaa tiedon tietokantaan.

```
public class EventManager {

   public static void main(String[] args) {
      EventManager mgr = new EventManager();

      if (args[0].equals("store")) {
         mgr.createAndStoreEvent("My Event", new Date());
      }

      HibernateUtil.getSessionFactory().close();
   }

   private void createAndStoreEvent(String title, Date theDate) {

      Session session =
HibenateUtil.getSessionFactory().getCurrentSession();

      session.beginTransaction();

      Event theEvent = new Event();
      theEvent.setTitle(title);
      theEvent.setDate(theDate);

      session.save(theEvent);

      session.getTransaction().commit();
   }
}
```

HibernateUtil-luokka ei ole Hibernaten mukana tuleva luokka vaan on ohjelmoitu tähän esimerkkiin helpoittamaan yhteyksien pyytämistä pääohjelmasta käsin. Yksinkertaisuudessaa se toimii Singleton-

suunnittelumallin kaltaisesti tarjoamalla saman istuntotehtaan kaikille sitä pyytäville.

4.3 Active Record vastaan Hibernate

Ruby on Railsin Active Recordia vertaamalla JBossin Hibernateen voidaan huomata, että Active Recordin käyttö on varsin helppoa ja vastaavasti Hibernaten käyttö työlästä. Hibernatessa tarvittava konfiguraatioiden määrä on suurempi verrattuna Active Recordiin Ruby on Railsissa. Ruby on Railsin tietokantayhteys konfiguroidaan muuttamalla config/-hakemistossa olevasta database.yml-tiedosta muutamaa riviä. Hibernaten tietokantayhteys konfiguroidaan XML-tiedostossa, joka on hankalampi ymmärtää ja muokata kuin Rails:n database.yml-tiedosto. Seuraava esimerkki on database.yml-tiedoston sisällöstä:

```
development:
  adapter: mysql
  database: fooga_development
  username: fooga
  password: ********
  host: localhost

test:
  adapter: mysql
  database: fooga_test
  username: fooga
  password: ********
  host: localhost

production:
  adapter: mysql
  database: fooga_production
```

username: fooga
*password: ********
host: localhost

Suurin ero kuitenkin Active Recordin ja Hibernaten välillä on se, että Ruby on Railsissa ei tarvitse konfiguroida oletuksena juuri mitään. Active Record löytää tietokannasta tarvittavat tiedot itse. Tämä perustuu Ruby on Railsin filosofiaan, jossa suositaan käytäntöjä konfiguroinnin sijaan. Tässä tapauksessa käytäntönä on nimeämiskäytäntö, jolla saadaan Active Record oletuksien avulla etsimään oikeata taulua tietokannasta. Hibernatessa on jokaisen luokan kohdalla konfiguroitava XML-tiedostoon tarvittavat parametrit, jotta Hibernate osaisi löytää tarvittavat tiedot tietokannasta. Projektin laajetessa myös XML-konfiguraatiotiedostojen määrä ja koko kasvaa merkittäväksi, kun taas Ruby on Railsin tapauksessa pysyttäessä nimeämiskäytännöissä ei juurikaan ylimääräisiä konfigurointeja synny.

5 VALIDOINNIT

Active Record Ruby on Railsissa tarjoaa työkalut tiedon validointiin. Tiedon validoinnilla tässä tarkoitetaan sitä, että tieto, jonka Active Record kapseloi, on oikeanlaista. Esimerkki validointitilanteesta on se, että user-model sisältää käyttäjänimen, joka on uniikki. Esimerkiksi Foogaan rekisteröityessä ei voida sallia käyttäjänimiä, jotka ovat käytössä, koska kirjautuminen tapahtuu käyttäjänimen mukaan kuten lukuisissa muissa web-palveluissa. Rails tarjoaa joukon valmiita validointimetodeita, joita ohjelmoija voi käyttää projektissaan. Tässä lyhyessä esimerkissä, joka on otettu Foogasta, käytettävä metodikutsu on seuraavanlainen:

validates_uniqueness_of :username

Jos tarvittavia validointimetodeja ei löydy suoraan Ruby on Railsin luokkakokoelmista, voi ohjelmoija ohjelmoida itse niitä lisää. Alimmalla tasolla ohjelmoija voi kirjoittaa toteutuksen yhdelle tai usealle seuraavista metodeista: *validate(), validate_on_create()* ja *validate_on_update()*. Kukin metodi määrittelee validointitoimenpiteet ActiveRecord-luokan elinkaaren eri vaiheissa. [12, s. 370] Valmiit validointimetodeja ovat [13, ActiveRecord::Validations::ClassMethods]:

- validates_acceptance_of
- validates_associated
- validates_confirmation_of
- validates_each
- validates_exclusion_of
- validates_format_of
- validates_inclusion_of
- validates_length_of
- validates_numericality_of
- validates_presence_of
- validates_size_of
- validates_uniqueness_of.

Seuraavassa koodiesimerkissä havainnollistetaan, kuinka voidaan itse määrittää modelille yleinen validointi sekä validointi luonnin yhteyteen [12, s. 371]. Yleistä validointia voidaan käyttää milloin tahansa kutsumalla olio.valid?, jolloin voidaan tarkistaa, onko olio validi. Tässä esimerkissä on validoitu käyttäjän nimen ainutlaatuisuus vain luonnin yhteydessä eli tarkoituksena ei ole enää nimeä sitä myöhemmin, joten sitä ei myöhemmin tarvita enää validoida. Koodiesimerkissä lauseke //-merkkien sisällä on säännöllinen lauseke, jonka avulla on tarkistettu, että nimessä kelpuutetaan vain alfanumeeriset merkit.

```
class User < ActiveRecord::Base
  def validate
    unless name && name =~ /^\w+$/
      errors.add(:name, "is missing or invalid" )
    end
  end
  def validate_on_create
    if self.find_by_name(name)
      errors.add(:name, "is already being used" )
    end
  end
end
```

Fooga-projektissa käytetään validointeja käyttäjäluokan luonnin yhteydessä. Oheisessa koodiesimerkissä nähdään, kuinka salasanavarmennus ja aikaisemmin mainittu käyttäjänimen ainutlaatuisuus on hoidettu validoinnilla. Myös salasanan pituus voidaan määritellä hyvin yksiselitteisesti. Koodiesimerkissä näkyy myös User-luokan suhteet muihin luokkiin. Liityntätekniikat ovat esitetty aikaisemmin kappaleessa 4.1.

```
class User < ActiveRecord::Base
  has_many :videos
  has_many :audios
  has_many :participants
  has_many :projects, :through => :participants
  has_many :versions
  has_many :notes

  validates_presence_of :username, :firstname, :lastname, :email,
:plain_password, :password_confirmation
  validates_uniqueness_of :username

  validates_length_of :plain_password, :minimum => 6, :message =>
@lang['dialogs']['login_short_password']
```

```
  attr_accessor :password_confirmation
  validates_confirmation_of :plain_password
end
```

6 YHTEENVETO

Tämä insinöörityö oli katsaus, millaisista osista Fooga-videoeditoria on lähdetty kehittämään. Käytettäviä teknologioita olivat: Ruby, Ruby on Rails, FFmpeg, Mencoder, ImageMagick ja FLVTool2. Työssä esiteltiin myös Fooga-videoeditorin hankkeen rakenne ja tulevaisuus (VIDEOS-hanke). Tavoitteena tässä työssä oli ohjelmoida Foogan perustoiminnallisuus ja tässä onnistuttiin hyvin. Puuttuvaksi ominaisuudeksi jäi koostetun videoprojektin imurointimahdollisuus järjestelmästä, mutta Fooga on kovan kehitystyön alla vielä tämän insinöörityödokumentoinnin jälkeenkin eli ominaisuuksia tulee viikoittain lisää.

Tässä työssä otettiin syvempi katsaus Object-Relational Mapping –maailmaan ja sen hyödyntämiseen Ruby on Rails -sovelluskehyksessä. Tarkastellaan myös, millaisia vastaavia tekniikoita Java-kehitysympäristöstä on käytettävissä. Vertailtiin myös Ruby on Railsin Active Record ORM-toteutusta ja JBossin Hibernate ORM-toteutusta keskenään. Uusin versio Javan EJB-tekniikasta (3.0) sisältää myös kehittynyttä ORM-toiminnallisuutta, mutta sen esittely päätettiin jättää tästä työstä pois. Lyhyesti esiteltiin myös Ruby on Railsin ActiveRecordin tiedon-validointimahdollisuuksia.

VIITELUETTELO

[1] Kokkonen, Juhana, *VIDEOS-Hankehakemus*. Helsinki 2006. Ei saatavilla.

[2] Eyespot. [verkkosovellus]. Saatavissa: http://eyespot.com/.

[3] Wikipedia, *Agile software development*. 10.3.07. [verkkodokumentti, viitattu 8.4.07]. Saatavissa http://en.wikipedia.org/wiki/Agile_software_development.

[4] Subversion, Subversion's features. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://subversion.tigris.org/.

[5] FFmpeg project, *FFmpeg Documentation*. [verkkodokumentti, viitattu 7.4.07]. Saatavissa: http://ffmpeg.mplayerhq.hu/ffmpeg-doc.html#SEC19.

[6] FFmpeg project, *FFmpeg Documentation*. [verkkodokumentti, viitattu 12.4.07]. Saatavissa: http://ffmpeg.mplayerhq.hu/documentation.html.

[7] MPlayer documentation. [verkkodokumentti, viitattu 12.4.07]. Saatavissa: http://www.mplayerhq.hu/DOCS/man/en/mplayer.1.html.

[8] FLVTool2. [verkkodokumentti, viitattu 12.4.07]. Saatavissa: http://inlet-media.de/flvtool2.

[9] Hunter, Timothy P, *User's Guide and Reference*. [verkkodokumentti, viitattu 6.4.07]. Saatavissa: http://www.simplesystems.org/RMagick/doc/usage.html.

[10] MySQL. [verkkodokumentti, viitattu 12.4.07]. Saatavissa: http://www.mysql.com/.

[11] Ruby-lang.org, *About Ruby*. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://www.ruby-lang.org/en/about/.

[12] *Interview with David Heinemeier Hansson from Ruby on Rails*. 2006. Saatavissa: http://dev.mysql.com/techresources/interviews/david-heinemeier-hansson-rails.html.

[13] Thomas, Dave ym., *Agile Web Development with Rails, Second Edition*. USA: The Pragmatic Programmers, LLC 2006.

[14] Rails Framework Documentation. [verkkodokumentti, viitattu 8.4.07] 2006, päivitetty 13.3.2007. Saatavissa: http://api.rubyonrails.org/.

[15] Forder, Justin, *ActionWebService*. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://wiki.rubyonrails.com/rails/pages/ActionWebService/.

[16] Heinemeier Hansson, David, *Secrets behind Ruby on Rails*. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://media.rubyonrails.org/-presentations/secretsofrubyonrails.pdf.

[17] Ruby on Rails Wiki, *Routes*. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://wiki.rubyonrails.com/rails/pages/Routes/.

[18] Hibernate – Overview. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://hibernate.javabeat.net/index.php.

[19] Object Relational Mapping Strategies. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://www.objectmatter.com/vbsf/docs/maptool/ormapping.html.

[20] Fowler, Martin, Active Record. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://www.martinfowler.com/eaaCatalog/activeRecord.html.

[21] Core J2EE Patterns - Data Access Object. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

[22] Hibernate, Chapter 1. Introduction to Hibernate. [verkkodokumentti, viitattu 8.4.07]. Saatavissa: http://www.hibernate.org/hib_docs/v3/reference/en/html/tutorial.html.